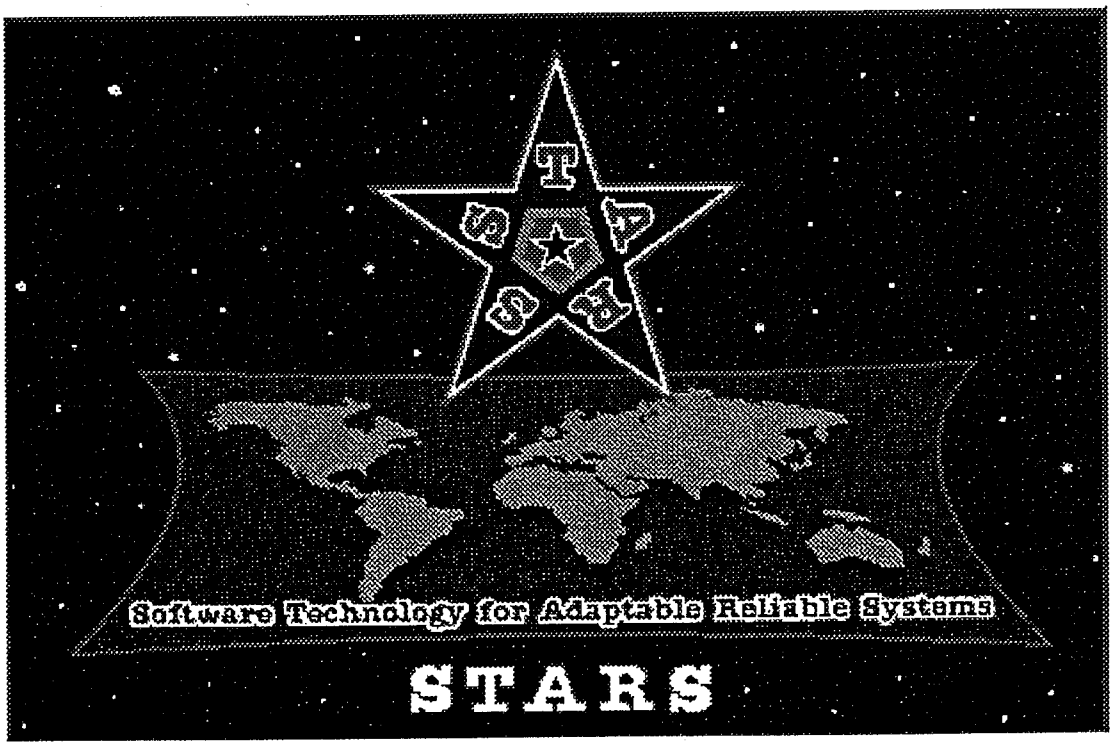
TASK: PV03
CDRL: A023
06 June 1994

Process Enactment Pilot Project Lessons Learned Second Interim Report

Informal Technical Data



STARS-VC-A023/009/00
06 June 1994

REPORT DOCUMENTION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	06 June 1994	Informal Technical

4. TITLE AND SUBTITLE
Process Enactment Pilot Project Lessons Learned Second Interim Report

5. FUNDING NUMBERS
F19628-93-C-0130

6. AUTHOR(S)
Ed Guy, Carol Klingler, Jim Baldo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091-3499

8. PERFORMING ORGANIZATION REPORT NUMBER
CDRL NBR
STARS-VC-A023/009/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
ESC/ENS
Hanscom AFB, MA 01731-2816

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report discusses preliminary observations on the integration of process enactment support technologies, resulting from the Unisys STARS Team Process Enactment Pilot project. A brief description of project plans and objectives is provided. Experiences to-date are highlighted in terms of the original objectives. Planning for additional activities is briefly addressed. This report is written to provide assistance to the Unisys STARS Team in their efforts to incorporate process enactment support into the software engineering environment (SEE) for the Army STARS Demonstration Project, to provide trial use feedback to the vendors of the tools being used, and to inform the STARS Program Office regarding activities to date. The observations reported may be of value to any organization attempting to integrate COTS tools in support of the automated enactment of engineering processes.

14. SUBJECT TERMS

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

TASK: PV03
CDRL: A023
6 June 1994

INFORMAL TECHNICAL REPORT

For

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Process Enactment Pilot Project
Lessons Learned
Second Interim Report

STARS-VC-A023/009/00
6 June 1994

Data Type: Informal Technical Data

CONTRACT NO. F19628-93-C-0130

Prepared for:
Electronic Systems Center
Air Force Systems Command, USAF
Hanscom, AFB, MA 01731-2816

Prepared by:
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: PV03
CDRL: A023
6 June 1994

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Lessons Learned
Second Interim Report

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A"of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0130, the STARS program is supported by the military services, SEI, and MITRE, with he U.S. Air Force as the executive contracting agent. The information identified herein is subject to change. For further information, contact the authors at the following mailer address:
delivery@stars.reston.paramax.com.

TASK: PV03
CDRL: A023
6 June 1994

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Lessons Learned
Second Interim Report

Abstract

This report discusses preliminary observations on the integration of process enactment support technologies, resulting from the Unisys STARS Team Process Enactment Pilot project. A brief description of project plans and objectives is provided. Experiences to-date are highlighted in terms of the original objectives. Planning for additional activities is briefly addressed. This report is written to provide assistance to the Unisys STARS Team in their efforts to incorporate process enactment support into the software engineering environment (SEE) for the Army STARS Demonstration Project, to provide trial use feedback to the vendors of the tools being used, and to inform the STARS Program Office regarding activities to date. The observations reported may be of value to any organization attempting to integrate COTS tools in support of the automated enactment of engineering processes.

TASK: PV03
CDRL: A023
6 June 1994

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Lessons Learned
Second Interim Report

Principal Authors():

Ed Guy	Date
Carol Klingler	Date
James Baldo, Jr.	Date

Approvals:

Teri F. Payton	6/14/94
Program Manager *Teri F. Payton*	Date

(Signatures on File)

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Phase II
Lessons Learned

Table of Contents

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Phase II
Lessons Learned

Table of Contents

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Phase II
Lessons Learned

Table of Contents

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Phase II
Lessons Learned

Table of Contents

1.0 Introduction

This report discusses preliminary observations on the integration of process engineering support technologies, resulting from the Unisys STARS Team *Process Enactment Pilot* project. A brief description of project plans and objectives is provided. Experiences to-date are highlighted in terms of the original objectives. Planning for additional activities is briefly addressed. This report is written to provide assistance to the Unisys STARS Team in their efforts to incorporate process engineering support into the software engineering environment (SEE) for the Army STARS Demonstration Project, to provide trial use feedback to the vendors of the tools being used, and to share experiences with other STARS program participants. The observations reported may be of value to any organization attempting to integrate COTS tools in support of the automated enactment of engineering processes.

1.1 Terminology

Feiler and Humphrey[1] define *process enactment* as "the execution of a *process* by a *process agent* according to a *process definition*". Throughout this report we have attempted to use the term in accordance with that definition; some ambiguity may be evident in certain contexts. Use of the term *execution* in this report implies automation, whereas *performance* may describe the activity of a human, a machine, or both.

A process *implementation* describes *how* process *steps* are performed, *how* performance of those steps is constrained, and *how* interaction between processes is coordinated. A manual process implementation combines all these process elements into a single process which is enacted as a unit. An *automated* implementation may separate these elements into an *enacting* process (which executes process steps) and one or more *control* processes (which apply constraints and govern interactions). The separate processes may be implemented at different times and may be enacted independently.

Process engineering support technologies include concepts, methods, and tools that assist the process *engineer* in the development of a process implementation, as well as those employed to assist the process *agent* in the performance of the process.

1.2 Objectives

Three primary objectives have been established for the project. The first is to **gain an understanding of existing and emerging process support technologies**. We have combined basic process engineering concepts and methods from the STARS program and other sources to construct a loosely connected process engineering framework. The "STARS Conceptual Framework for Reuse Processes (CFRP)" and the "STARS Process Concepts Summary" contribute to the framework, along with *Structured Analysis & Design Technique (SADT)* and a *Process Breakdown Structure* formalism. It is not our intent to test or formally verify any particular aspect of the

---

1. Feiler, Peter H. and Humphrey, Watts S.
Proceedings of the Second International Conference on the Software Process
Berlin, Germany
February 25 - 26, 1993

framework, but to determine how well the concepts relate to actual practice and whether the methods promote consistent analysis.

The second major objective of the project is to **gain expertise in the application of the tools** selected as candidates for integration into the STARS SEE. The kinds of expertise needed varies, depending on the *role* occupied by a particular user of the tool. *Process engineers* need to understand the specific capabilities and limitations of each tool before applying the tool to a task. They need to know what skills will be needed to apply the tool and the level of effort required. A corresponding knowledge of how to access tool capabilities, avoid limitations, acquire skills, and estimate effort is needed by *process performers* who will use the tool. *Environment integrators* need to understand the mechanisms provided by the tool for exchanging information with other tools and with external processes.

In the construction of a *process-driven* environment, expertise in the use of a tool as an isolated set of capabilities may not be adequate to provide an acceptable level of process support. It is also important to understand how those capabilities can be combined to enact a prescribed process. Often, a capability in one candidate tool is duplicated, in part or in whole, in another candidate tool. Such duplication can, potentially, introduce process conflicts or inconsistencies in the exchange or interpretation of data and control information, as well as add complexity to the presentation of the target process. Tools designed to implement a specific process may perform well in the execution of that process, but be difficult to adapt to other applications.

Some tool providers attempt to avoid imposing a particular process upon the users of their products. Absent a formal concept of operations, these tools often reflect an *apparent application paradigm*, stemming from their developers' implicit or assumed models of use, that can be difficult to characterize. The level of consistency in the tool's representation and execution of that paradigm can impact the tool's ability to effectively communicate its intended mode of operation to a user, the ability to integrate with the capabilities of other tools, the internal interoperability of the tool's capabilities and the ability to apply those capabilities to support a particular process. Environment integrators need to understand the apparent application paradigm of each tool. They need to determine the degree to which the paradigm is compatible with target processes, whether or not the tool can support alternate models of use, and the effort required to enable that support.

The third objective of the project is to **enable process engineering support for the Army STARS Demonstration Project.** The Army CECOM Software Engineering Directorate is participating with the STARS program, applying new tools and technologies to support improvements in the processes used to maintain and evolve software for mission-critical systems. The knowledge and expertise gained from the Process Enactment Pilot project will help to increase the Demonstration Project's confidence in the adoption of process enactment technologies. The application of validated concepts and methods can assist in the analysis and definition of software development processes, and in the identification of enactment roles to be satisfied by the SEE. Tool expertise can assist in the selection of individual tool capabilities to satisfy those roles, to ensure mutual support among the capabilities, and to circumvent any incompatibilities that may exist. The Process Enactment Pilot project's characterizations of requisite skills and levels of effort needed for process engineering activities can assist the Demonstration Project in planning its own process enactment and training efforts. The combined experiences of the Process Enact-

ment Pilot and the Demonstration Project may provide enough practical experience to determine the feasibility of defining a generic *process implementation process*.

1.3 Requirement Assertions

During initial planning various issues were raised by the process engineering team concerning approaches to process enactment being pursued by different STARS organizations, potentially negative effects of process automation, the reliability of available technologies, etc. Throughout the experiment these issues have continued to guide and constrain the direction of the project. In effect they form a set of informal assertions of requirements that govern the effort.

1.3.1 Process-driven Environments

Assertion

> The operation of an automation environment should be governed by the properties of the processes enacted within the environment, not by the properties of the mechanisms employed to enact the processes.

Rationale

The goal of any process automation effort should be to allow humans to make more effective use of their analytical and creative skills by minimizing the tedium associated with the performance of repetitive tasks. All too often, the automation of a work process has failed to achieve this goal, causing many individuals and organizations to resist the idea of process automation. In many cases the failure can be attributed to the methods used to implement the process.

Automated implementations of processes can tend to be driven more by the capabilities of the tools used to automate the processes, than by the processes being automated. Absent a high degree of compatibility between a tool's application paradigm and the human agent's process paradigm, a significant conceptual reorganization of the process can occur during development of the implementation. Rather than provide automated mechanisms to assist the agent in the performance of a task, the implementation may present a completely different task to be performed. The new task may bear little resemblance to the work with which the agent is familiar and may require application of a radically different set of skills, making the automated process more difficult to perform than its human-enacted counterpart. The implementation may require assistance from the human agent to support its internal processes, thereby reversing the roles of the agent and the environment, increasing the amount of tedium imposed upon the agent, and making less effective use of the human resource.

The behavior of any process is inherently dependent upon the general capabilities of and specific mechanisms employed by the agents that perform the process, including those of both automated tools and human agents. The application of any particular set of automated mechanisms may necessitate radical changes in the internal organization of a process task. Such changes should not be allowed to fundamentally alter the human agent's concept of the work to be done.

1.3.2 Extra-environmental Activities

Assertion

A process implementation employing automation to execute process tasks must provide coordinate support for process tasks that are performed outside the context of the automation environment.

Rationale

Large-scale process automation has proven somewhat ineffective due to a necessary concentration of attention on environment-centered processes. Many of the tasks associated with a software development effort are not performed by automated mechanisms, making it difficult to support those tasks in an automated environment. There is a need to develop effective techniques for supporting unobtrusive interaction between manual and automated performance mechanisms.

1.3.3 Effectiveness of Enactment Support Technologies

Assertion

To be of significant value, technologies employed in the development of automated processes should offer significant advantages over traditional software development technologies.

Rationale

Development organizations possess the requisite knowledge and skills to apply software engineering techniques to solve technical problems. Software development technologies are being developed to reduce duplication of effort in the development of reusable software systems. Implementation of automatically enactable processes can be accomplished using the same knowledge, skills, and technologies. Such development can, however, be performed by individuals who are not grounded in traditional software development methods if process engineering technologies can minimize dependency upon the individual's software engineering skills. Process engineering technologies that fail in this regard may offer little added value over that of other technologies supporting software development.

1.3.4 Incremental Process Enactment

Assertion

Support for incremental process automation techniques is a consideration in the recommendation of candidate technologies.

Rationale

An organization may have neither the resources nor the skills needed to undertake comprehensive re-engineering of their business processes prior to initiating process automation efforts. An organization may also consider it undesirable to adopt and implement externally defined processes or

process segments requiring radical changes in their fundamental practices. An organization may approach automation of process enactment incrementally, perhaps in conjunction with a general process improvement program.

1.3.5 Tool Selection Restrictions

Assertion

The set of candidate tools and technologies for the Process Enactment Pilot project is limited to those which are currently available from commercial sources, for which prototypes are available, or are under development by a STARS participant. STARS participants, particularly Unisys STARS commercial partners and Prime Affiliates, are afforded special consideration in the selection of candidate technologies.

Rationale

A typical process engineering effort will be constrained in its selection of tools and technologies. Organizations typically do not have the resources needed to develop their own automation environment to address their process needs, nor are they willing to accept the risk of using research prototypes or unsupported freeware in production environments; acquisition of technologies from commercial sources is necessary. Furthermore, contractual agreements, special purchasing arrangements with either internal or external suppliers, and budgetary restrictions may limit the commercial technologies an organization can apply.

In recognition of these practical realities, STARS is chartered to apply and integrate available commercial products that support the megaprogramming paradigm and to accelerate the transition of new technologies into the commercial marketplace in areas that current commercial offerings do not adequately address. The Unisys STARS team has formed alliances with several organizations, including Hewlett-Packard, Software Design & Analysis, and Amadeus Software Research to help achieve these objectives. Commercial technologies from each of these companies are being integrated and applied in the Process Enactment Pilot project. In fact, the Unisys STARS team was instrumental in transitioning the Amadeus research prototype into a commercial product.

2.0 Approach

The objectives of the Process Enactment Pilot project and the requirement assertions developed by the engineering team describe an approach to the experiment that balances the needs of the human process agent with the capabilities of the available tool set. Development of a detailed process description will occur in conjunction with the accumulation of tool expertise. The following activities have been selected to support this approach:

- Select a process for pilot enactment
- Identify a set of candidate enactment tools
- Capture a definition of the existing process
- Identify tool capabilities to support process roles
- Encode the process in enactable form

- Automate process steps using individual and integrated tool capabilities
- Perform the process in an operational context
- Evaluate the effects of automation on performance of the process
- Adapt the process to take advantage of additional capabilities within the environment

2.1 Selection of a Pilot Process

Four general criteria for the selection of a pilot process are defined; other considerations, reflecting known limitations or advantages of a particular tool set could be used to augment these criteria.

1. The process must be currently in use. Experimentation with a *live* process provides for trial enactment of the process in an operational environment and comparison against the existing implementation.
2. The process must be well understood. Selection of a process familiar to the team allows the team to concentrate its efforts on the application of candidate technologies, minimizing the need for process training.
3. The process must be reasonably complex. That is, it must have readily identifiable bounds, so the scope of the effort can be contained, yet be complex enough to exercise the technologies.
4. The process must provide an opportunity to apply the knowledge gained through the experiment and measure the impact of its application.

Our own internal STARS document delivery process was selected for the project. This process involves the submittal, review, and delivery to various destinations of STARS software and documentation, technical reports, etc. and is used daily. While written descriptions exist for some activities, most of the process knowledge rests in the experience of the STARS Data Manager. Automated support for some activities is already provided via Unix shell scripts. The pilot project team shares a common understanding of the purpose of the process and the activities involved. The process is rather broadly defined, yet consists of a number of relatively simple activities; it involves multiple user roles, collateral activities, and non-sequential dependencies. Performance of the process has been subject to significant productivity variations, induced by changes in both personnel and operating environment. Figure 1 depicts a top-level description of the process.

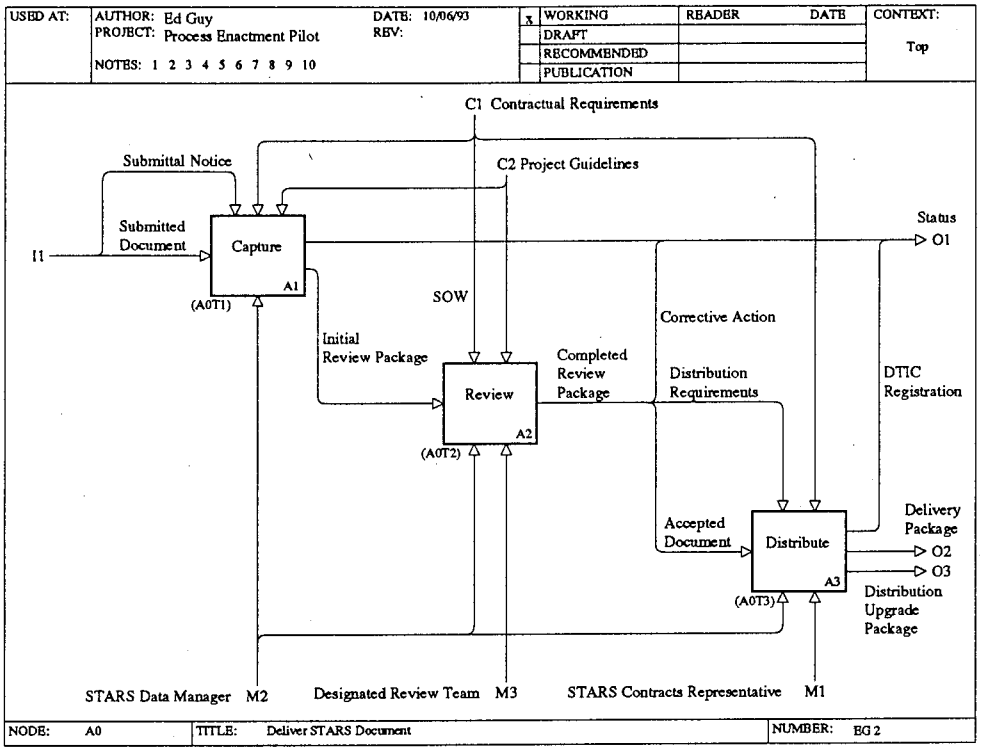


Figure 1
Deliver STARS Document - SADT Diagram

2.2 Candidate Tools

2.2.1 Amadeus Measurement System

Amadeus Software Research
Irvine, California

Version: 2.1.7
Platform: Sun Sparc, HP 9000/700

Amadeus is a measurement-driven analysis and feedback system that provides infrastructure and services to automate the collection, analysis, and visualization of process and product metrics. The tool is a commercial evolution of an initial prototype developed by Dr. Richard Selby of the University of California at Irvine, as part of the ARPA-funded Arcadia project. Amadeus provides a low entry barrier to measurement with dynamic, unobtrusive, tailorable capabilities for monitoring processes, tools, and interactive user sessions. Mechanisms for interactive and programmatic capture of numeric and symbolic measurement data are provided, including application programming interfaces for well-established high-order programming languages, Unix shell interpreters, and the SoftBench Broadcast Message Server.

2.2.2 AutoPLAN

Digital Tools, Inc.
Cupertino, California

Version 1.0.1
Platform: Sun Sparc

AutoPLAN is an interactive project scheduling and control package for Unix-based systems, designed for distributed processing environments. Various graphical representations are supported, including Gantt charts, CPM diagrams, and Work Breakdown Structures. Its built-in report generation capabilities provide support for time, resource, and cost analysis, as well as progress monitoring.

2.2.3 Design/IDEF

Meta Software Corporation
Cambridge, Massachusetts

Version: 2.5.3
Platform: Sun Sparc

Design/IDEF is a process modeling tool that supports generation of graphical process representations. Capabilities for producing IDEF0 activity diagrams, IDEF1 and IDEF1X data diagrams, and Entity-Relationship diagrams are provided, along with method-independent drawing capability. IDEF0 modeling reflects hierarchical task structures, data availability requirements, process control influences, and selected performance mechanisms; IDEF0 diagrams do not dictate specific activity sequencing. Design/IDEF's IDEF0 models can be augmented with activity-based costing information. Work product modeling is supported by the product's various data representations. Companion products Design/CPN and WFA Simulator provide support for various aspects of work flow analysis.

2.2.4 MSP & PM Tool Sets

Innovative Software Engineering Practices Inc.
Leesburg, Virginia

Software Design & Analysis Inc.
Boulder, Colorado

Version: 2.2
Platform: Sun Sparc / PCTE

The Minimally Structured Process (MSP) and Process Mapper (PM) Tool Sets provide process description and analysis capabilities built around the Process Breakdown Structure (PBS) formalism. MSP supports description of the intent and work involved in performing process steps, decomposition and control relationships among process steps, and the data availability relation-

ships between process steps and work products. It assists in the accumulation of knowledge about work processes through automated analysis of the syntactic correctness, referential integrity, completeness, structure, and consistency of a description of the processes. PM extends these capabilities through more formal descriptive techniques and by supporting definition and analysis of work product states, sub-step sequencing, roles and communication obligations.

2.2.5 PCMS (Product Configuration Management System)

SQL Software Ltd.
Vienna, Virginia

Version: 2.0
Platform: Sun Sparc

PCMS supports management of software product configurations, problem reporting and change control activities, baseline management, release management, and authorization control. PCMS provides for the development of hierarchical product structures against which various version control techniques can be applied. Tool features support the definition of product variants, change control documents, and product life-cycle management roles. Rules can be defined to relate change documents to different kinds of managed items and for constructing products from managed components. Automatic generation of formal and informal product baselines is supported, along with the definition and control of product releases. Many capabilities of PCMS can be applied to the management of products whose components do not reside within the automation environment (printed documents, hardware, training materials, etc.).

2.2.6 SynerVision

Hewlett-Packard, Software Engineering Systems Division
Fort Collins, Colorado

Version: A.00.01
Platform: HP 9000/700

SynerVision is a process execution tool that supports individual and group task management, automation of task execution, hierarchical task measurement, and status reporting. Personal and multi-user shared projects can be created. Reusable process templates can be constructed for instantiation with project-specific attributes and actions. The tool provides for both textual and graphical presentation of task hierarchies in a user-controlled viewing configuration. Sequential dependencies among tasks can be defined for use in either an advisory or enforcement capacity. SynerVision is available for HP 9000 series platforms, with Sun Sparc support under development. It is fully integrated with the SoftBench Broadcast Message Server. Automated actions can be executed via SoftBench messages and Bourne shell programs.

2.3 Tool Integration Strategy

Figure 2 represents one organization of the candidate tools envisioned by the team. Not all of the capabilities described are currently enabled within the tool set, but they are considered technically

feasible and reasonably compatible with the operational paradigms of the individual tools. This arrangement has been of value to the process pilot team in the automated enactment of an existing process. Other organizations have successfully applied the same tools in different relationships to support development of new processes. Functional overlap among the tools and the variety of operational paradigms represented should allow integration of the tools to be tailored to the needs of the users.

In this particular arrangement, MSP & PM would be used to produce a formalized Process Breakdown Structure (PBS), which would provide a basis for the construction of IDEF0 diagrams and other process/product models by Design/IDEF or a similar tool. Such models would be suitable for training, process improvement activities, etc. General task structures and resource requirements would be extracted from the PBS and passed to AutoPLAN for development of a project management plan. Detailed, step-by-step task definitions and role responsibilities would be extracted from the PBS for development of project-specific SynerVision task hierarchies and automated actions, which would be instantiated based on a Work Breakdown Structure (WBS) and corresponding schedule produced by AutoPLAN. The PBS would also supply information about work products, suitable for construction of PCMS product structures and control plans.

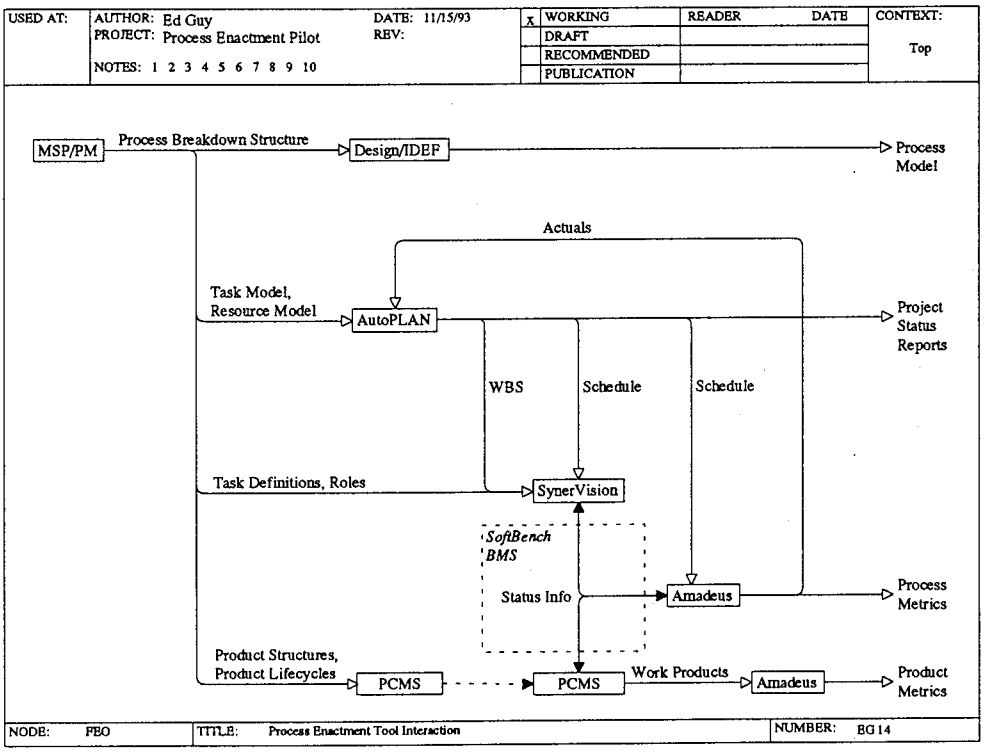


Figure 2
Tool Interaction Scenario

In general, activities would be initiated by SynerVision tasks which would transfer control to PCMS whenever the activity involved construction or modification of a component represented in the PCMS product management structure. PCMS would invoke Amadeus measurement agents to compute, record, analyze, and report product metric information, and would transfer control back

to SynerVision upon completion of its product-oriented activity or when instantiation of additional task hierarchies is needed. Communications between these two tools would be accomplished by exchange of messages via the SoftBench Broadcast Message Server. These exchanges would be monitored by the Amadeus measurement system, which would record information about process and product states, relative to the baselined schedule, and pass actual completion dates, elapsed time, etc. to AutoPLAN for generation of project status reports and for interactive management analysis.

3.0 Lessons

3.1 Capturing the Existing Process

Based on our knowledge of the process and available resources, the engineering team initially decided not to generate any formal models of the existing process. We believed we could develop a reasonable task skeleton, representing a complete decomposition of the process from its highest level of abstraction to its simplest individual process step. Once this had been accomplished we would begin to address dependencies, work product interactions, etc. We began building a process description by interactively entering a SynerVision hierarchical task structure. Since the process is relatively simple and partially automated we expected to be able to generate a reasonably complete, perhaps executable representation using SynerVision's basic capabilities. We learned quickly that a simple hierarchy of steps can represent only a small portion of the information needed to adequately describe a process. We noted that each level of decomposition tended to raise as many questions as it answered, effectively communicating *what* the process steps were, but not *why* they were being performed. It was easy to become absorbed in the details of process execution that did more to obscure the intent of the process than to expose it.

In some respects, our intimate knowledge of the existing process was proving inappropriate to the needs of automation. Many tasks translated readily to a SynerVision task hierarchy, however, reproducing the same sequence of detailed steps that had been performed manually often introduced an unnecessary layer of computerized intervention between the human agent and the goal of the process. Many such task sequences manipulated low-level process support artifacts (forms, routing sheets, reports, etc.) that provided assistance to the human agent in performing the process, but which might not be necessary in an automated environment. Such detailed tasks were identified as candidates for automated execution.

Conversely, much of our process knowledge did not readily translate to a hierarchical structure. A human agent could quickly assimilate the location and organization of a bookshelf containing several years of contractual correspondence, letters of agreement, notes on internal procedures, etc. for reference when needed. Some sort of automated support was needed to ensure consistent application of the information represented on that bookshelf. The decision-making processes in which that information is used, though trivial for a human agent, introduced an unexpected level of complexity into our automated process. It became necessary to adopt a more disciplined approach to development of an initial process representation and to identify mechanisms for representing the process in more formal terms.

3.1.1 Establishing Task Granularity

Given our difficulties with hierarchical analysis, we attempted to establish a logical basis for decomposition of task structures. SynerVision documentation suggested that a distinction between task and action might be useful. (SynerVision provides for the association of manual and automated actions with each process task). While this distinction may prove useful in the execution of the process in a SynerVision implementation, it added no additional value in our attempts at producing a general description of the process. Other equally unsatisfying distinctions (what vs. how, problem vs. solution,...) were discarded. We surmised that an effective level of granularity might be achieved by examining the task steps in relation to the physical objects (files, forms, prints) upon which they operated. Unfortunately, these object relationships seemed even more difficult to characterize than those between the steps.

3.1.2 Process Modeling

3.1.2.1 Activity Modeling

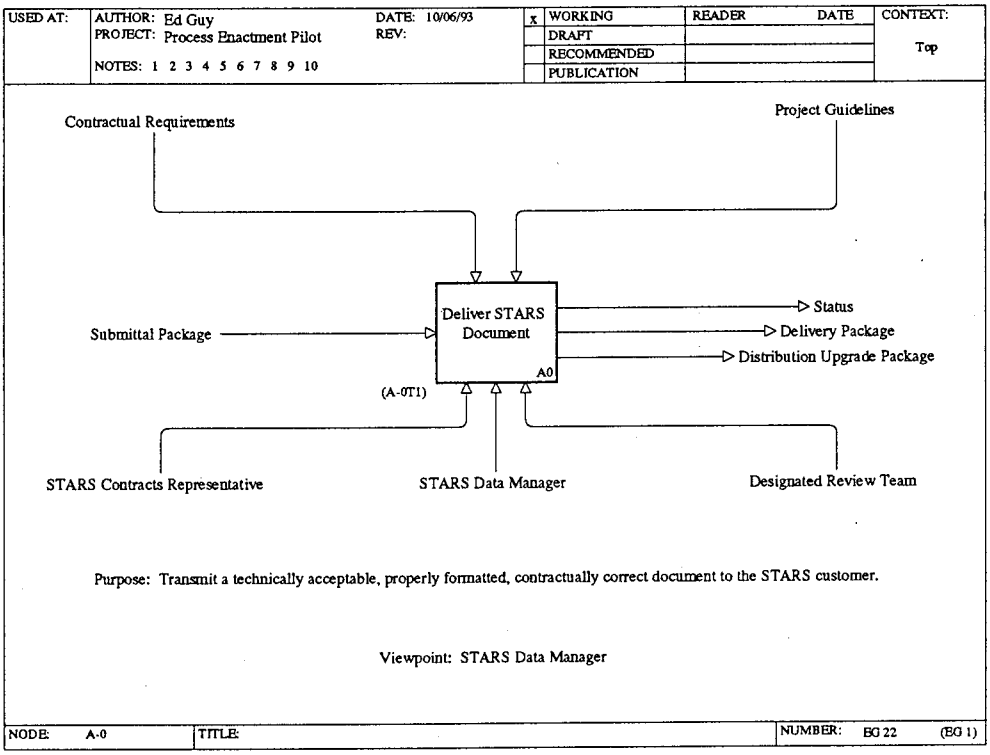


Figure 3
SADT Context Diagram

The necessity to prepare a more complete, more formal model of the process became apparent. This re-energized interest in SADT modeling techniques (Figure 3). SADT modeling (the basis for IDEF0) can produce a rich, expressive representation of complex process interactions and can quickly expose errors and omissions in the process model, provided the needed information is available and properly organized. The amount of information needed depends upon the complexity of the process and the number of agents involved in its performance, and whether the model is

to represent an existing process or one which is being defined for the first time. For initial definition of a process SADT diagrams can be used effectively to develop a top-down decomposition of the process, reflecting increasing detail as the process evolves. For description of an existing process, an abstract view of the process is typically derived from detailed information about how it is performed. Completeness and consistency of that information is critical to the development of an accurate graphical abstraction. Unfortunately, beyond some techniques for interviewing process experts, the SADT method does little to guide the process engineer through the collection and categorization of such information.

3.1.2.2 Process Breakdown Structures

Our Prime Affiliate, *Software Design & Analysis* (SDA), provided useful guidance in organizing the process knowledge being collected by describing a process analysis approach based on *Process Breakdown Structures* (PBS). Our application of this approach began with recording process knowledge in terms of the *essential properties* of the process. We recorded the *purpose* of, *entry and exit criteria* for, *work products used and generated* by, and a textual description of the *activities* performed within a task or sub-task (see Figure 4) with little immediate concern for its context. Once the initial information had been recorded for a task, its parent, and its immediate children, a few quick iterations through the structure, adding annotations and adjusting descriptions, produced an understanding of the process that was adequate to support construction of a preliminary SADT model. Examination of that preliminary model exposed problems and inconsistencies that were addressed either by re-examination or further decomposition.

USED AT:	AUTHOR: Ed Guy PROJECT: Process Enactment Pilot NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 10/06/93 REV:	x	WORKING	READER	DATE	CONTEXT: Top
				DRAFT			
				RECOMMENDED			
				PUBLICATION			

Purpose:
Deliver a technically acceptable, properly formatted, contractually correct document to the STARS customer.

Description:
The STARS Data Manager verifies accessibility of the Submittal Package, the reproducibility, and traceability to the Contract Data Requirements List (CDRL) of the Submittal Package, acknowledges receipt, and routes the document for review by a Designated Review Team. The team reviews the document content for compliance with the Contract Statement of Work (SOW) and Contract Data Requirements List (CDRL). If the document is accepted, the Data Manager delivers print and electronic representations of the document to the recipients identified in the CDRL, forwards a copy to ESC for distribution upgrade (if required), and registers the document with the Defense Technical Information Center (DTIC). If the document is rejected the Data Manager notifies the submitter and arranges for subsequent resubmittal.

Entry Criteria:	Activities:
Submittal Announced	Capture Submittal Package
	Review Submitted Document
Work Products Used:	Distribute Accepted Document
Submittal Package	
Work Products Generated:	
Delivery Package	
Distribution Upgrade Package	
Status	
Exit Criteria:	
Submittal Rejected	
Document Rejected	
Delivery Verified	
Distribution Upgrade Announced	
Distribution Upgrade Denied	
DTIC Registration Announced	

NODE: Text A-0T1	TITLE: Deliver STARS Document	NUMBER: BG6

Figure 4
A-0, Text Page 1: Process Breakdown Structure

3.1.2.3 Work Product Modeling

Due in large part to a lack of appropriate skills and, partially, to a particular difficulty in aggregating process work products in meaningful ways, very little was done to represent the structure and interrelationships of the work products associated with the target process. We have been able to identify three categories of work products for our process:

- *primary* - the work product whose manipulation is the purpose of the process - in our case, the *STARS Document* to be delivered
- *companion* - electronic delivery notices, letters of transmittal, and other work products that represent the primary work product in a corollary process
- *support* - forms, routing sheets, reports, and other work products that are used to manage the process itself

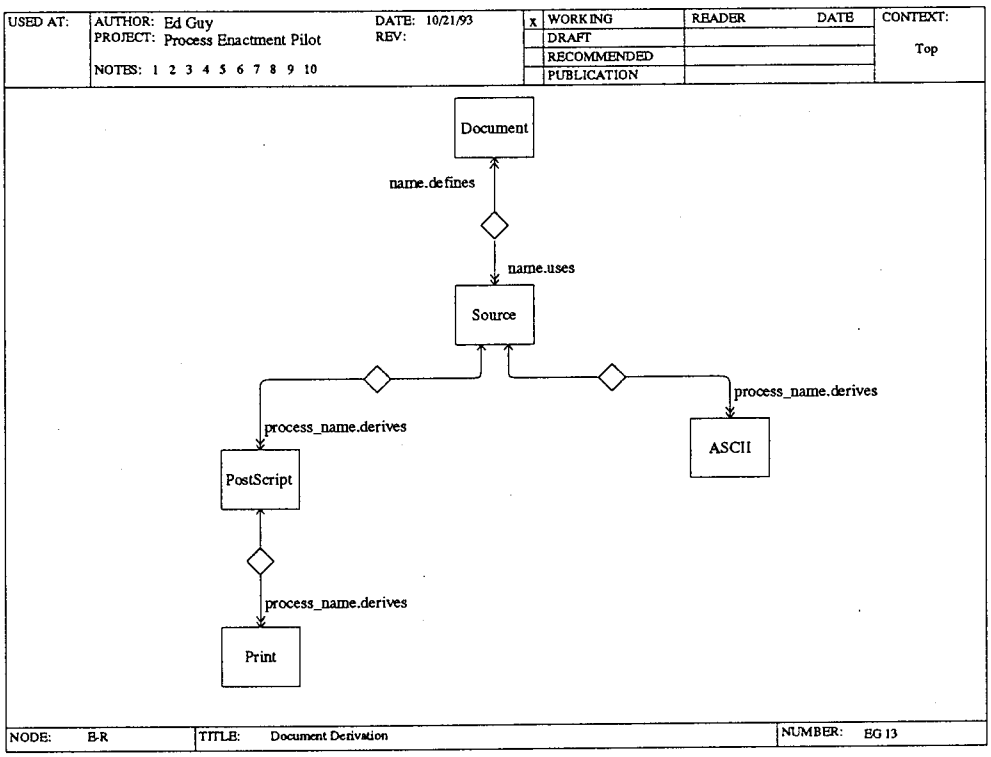


Figure 5
Primary Work Product Derivation

A simple derivation tree (Figure 5) was produced for the *primary* work product of the target process, and was useful for defining a product build sequence within PCMS. Our current tool set does not provide adequate support for modeling work products and their relationships to each other. Perhaps, a good ERA diagramming tool would be helpful.

3.1.2.4 Life-cycle Modeling

We also generated a *life-cycle model*, consisting of an interlaced set of simplified state-transition diagrams (Figure 6) showing pertinent portions of work product life-cycles and their relationships to each other. Our intent was to use these diagrams to define *control plans* for PCMS, which proved inappropriate. The diagrams have helped to identify natural boundaries in the structure and flow of the target process. A consistent method for representing interrelated product life-cycles is needed.

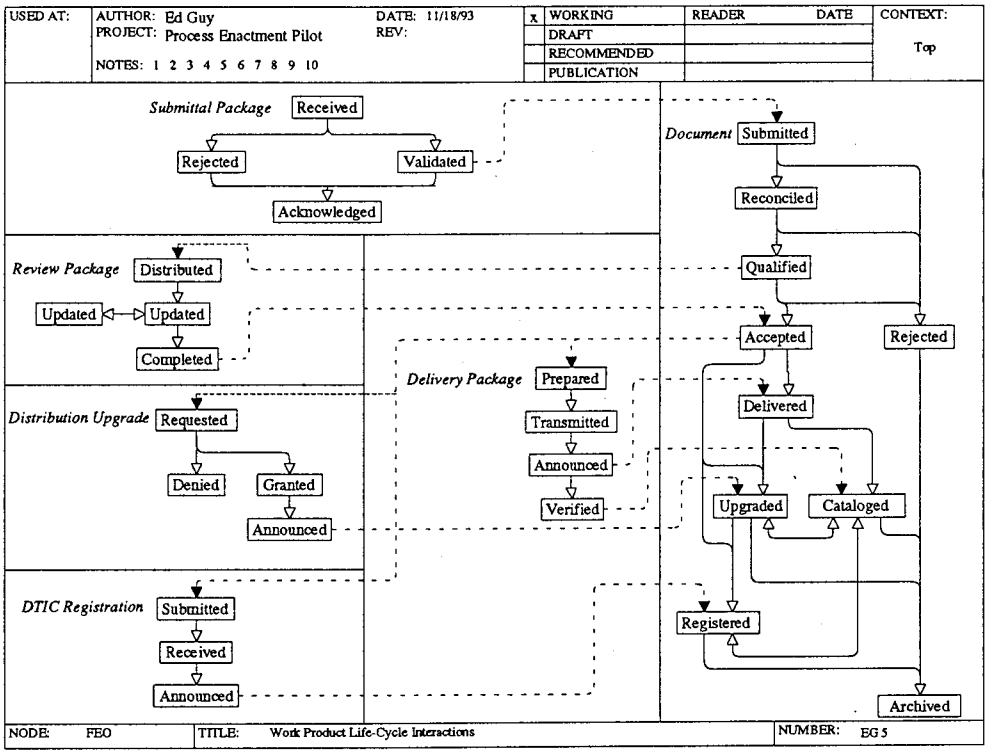


Figure 6
Simplified Work Product Life-Cycle Model

3.1.2.5 Adjacent Abstractions

Whatever sort of analysis may be in progress, whether Functional (activities), Informational (work products, for now), or Behavioral (represented by our life-cycle models) an interesting pattern has been noted. Whatever the subject of analysis may be, that subject cannot be reasonably understood without a working model of its *adjacent abstractions*. Before one level of decomposition can be completely defined, it is necessary to construct a preliminary model of the next lower level, which raises questions about the subject level, which can only be answered by analysis of its higher-level abstraction. Similarly, before a level of decomposition for an activity can be completely defined, it is necessary to construct preliminary models of work-products and life-cycle interactions related to the activity. These models might be viewed graphically above, below, and to either side of the subject model and are, therefore, considered *adjacent*.

3.1.2.6 Rapid Iteration

This iterative analysis of adjacent abstractions is typically repeated several times until an adequately defined, consistent model view has been produced. The more times the cycle is repeated, the more complete and consistent the view becomes. There is a tendency to define one abstraction as completely as possible before examining adjacent abstractions. Experience suggests that the rate at which the desired level of completeness and consistency is achieved may be more closely related to the number of iterations than to the amount of time spent in each. By producing a consistent, perhaps shallow, view of each adjacent abstraction during one iteration, it should be easier to maintain that consistency during subsequent iterations, thereby reducing the overall cycle time.

3.1.2.7 Tool Selection Influences

Enactment tool architectures and capabilities had less influence on our process modeling and analysis activities than originally expected. Certainly, there is nothing in the SADT activity model that reflects particular SynerVision capabilities. Tasks have been defined with a formal hierarchy in mind, and our knowledge of SynerVision capabilities has affected the level of detail addressed during analysis. There is no evidence, however, that the result would have been different if any other task management tool were targeted.

Since an understanding of the process work products was needed, since work product modeling was incomplete, and since PCMS uses state transitions to govern product life-cycles, a PCMS-style life-cycle model was produced for the work product set. We expected that model could be used initially to specify a PCMS control plan for the primary work product of the process. Lack of flexibility in the definition of PCMS control plans has inhibited satisfactory instantiation of the life-cycle specification within the tool; alternate means of representing the specified life-cycle in PCMS are being investigated. It has been necessary to revise the life-cycle model somewhat to ensure compatibility with the PCMS paradigm. In this particular instance, the modification does not significantly alter the human agent's view of the process being performed. Under other circumstances such incompatibilities may necessitate more formal trade-off analyses, comparing the impact on the human agent to the effort required to maintain an existing operational paradigm.

It appears that particular tool capabilities may have little effect on the development of an abstract *process model* that describes a target process. It should be practical to continue evaluation of candidate tools until the abstract process model is reasonably complete. The effect of tool selections should not become apparent until a particular implementation of the process is constructed.

3.1.3 Summary

Originally, the team planned to build a skeleton SynerVision task hierarchy for the entire process, based on the granularity of the existing manual task hierarchy, then define the essential properties for each task in the structure. In practice, it has been virtually impossible to decompose the task structure even one level without a good understanding of its essential properties, so definition of the task hierarchy has been driven by analysis of the essential properties, not the other way around. We have also learned that a process-independent view of process work products can provide valuable insight for organizing development activity. Knowledge gained through product life-cycle modeling activity helped to define potential integration points among process tasks and

tools - that is, by developing alternate abstractions of the work product structure, we were able to identify natural boundaries (primary work products, companion work products, and supporting work products) that supported an iterative integration strategy. Development of a descriptive process model, whether representative of an existing process or of a process to be implemented, can be completed successfully without knowledge of the tools to be used in the automation of the process. The capabilities of particular tools do, however, affect the development of a particular implementation of the process.

3.2 Applying Tool Capabilities

The majority of the engineering team's effort has concentrated on the manipulation of a particular instance of the primary work product of the process, the *STARS Document*. The principal environmental roles to be supported are those of *Task Manager* (satisfied by SynerVision), *Product Manager* (PCMS), and *Metrics Manager* (Amadeus). Although AutoPLAN and Design/IDEF do not play significant roles in performance of the target process at this time, they have been employed to assist with project planning and process modeling efforts.

This section summarizes the successes and failures of the team in application of the candidate tools to their designated roles. The tools have proven to be generally useful and to adequately satisfy their enactment roles. In general, continued use of a tool reflects overall confidence in its long-term prospects. Negative findings may reflect a lack of understanding on the part of the team or an incompatibility between the team's application of the tools and the intent of the tool developers, or may expose an inadequacy in utility or quality of the tools. More detailed observations are provided as appendices to this report.

3.2.1 AutoPLAN

As mentioned in section 3.3 "Tool Integration Strategy"our long-term vision includes export of AutoPLAN work breakdown structures for translation into SynerVision task structures and *vice versa*. This potential capability has not been investigated. We have, however, used AutoPLAN successfully to help plan and organize the project, providing an overall view of our activities which may serve as a basis for definition of a process engineering project model. The tool's user interface is generally consistent, though not always intuitive. With a little experience most of the user interface quirks become minor distractions. Generation of printed graphs and reports is cumbersome and deserves some attention from the developer. The results of resource analysis and schedule calculation algorithms aren't always predictable and unusual scheduling situations can cause the tool to crash. Product documentation has not served as an adequate reference for trained users. Project data is stored in a proprietary database, accessible only via the tool's user interface. An available (but currently unsupported) C language Application Programming Interface reportedly supports such major operations as database query and update, display of charts and tables, generation of reports, and invocation of schedule calculation algorithms. For typical project planning and tracking activities, most users should find the tool easy to use with minimal training or with initial assistance from an experienced user. Applicability of individual capabilities in a process-driven environment will depend upon accessibility provided by the API.

3.2.2 Design/IDEF

In many ways Design/IDEF has been a truly valuable tool. As shown in this report the tool supports representation of a variety of information, reflecting its origins as a general purpose drawing tool. We have been able to consolidate several different kinds of information in a single *model*, suitable for iterative analysis. Some tolerance, persistence, and flexibility are required on the user's part to achieve acceptable results. The tool is effectively closed to its environment, providing no external access to its database. A complete model is represented in a single Unix file, which defines the level of granularity for access. While implementing the necessary mechanics to draw IDEF0 activity diagrams, the tool provides no support for author/reader cycles, version management, or other important aspects of the IDEF0(SADT) method. To produce effective IDEF0 models, users should be well versed in IDEF0 modeling techniques. Due to inaccessibility of the underlying database, it is not practical to use the tool to execute processes interactively. If models are kept small the tool can be used to provide interactive graphical guidance to users of a process that is driven by other means. We have experienced reliability problems with the Unix version of the tool, which appear to have been resolved on other platforms (PC, Macintosh). The company acknowledges its reduced level of support for Unix users. This situation introduces an element of risk into our process modeling activities.

3.2.3 SynerVision

SynerVision occupies the *Task Manager* role in our process automation environment. It provides the primary interface between the human agent and the tool set. We have created a simple task structure that outlines some typical categories of activity (for example, "Maintain Project Plan", "Handle Correspondence", "Prepare CDRLs for Delivery") to be performed in a project context. By selecting one of these categories for execution the agent enables a pull-down menu of actions pertinent to the selected task. Selection of a menu item may initiate execution of an external program, instantiate a pre-engineered process template to create a detailed activity hierarchy for the executing task, invoke capabilities in other tools, transmit process information to the environment via a BMS message, or present textual or graphical guidance for manual performance of the task.

The effectiveness of SynerVision's user interface presentation depends on how well one of the tool's models of use (see section 3.2.3.1) maps to that of the process performer and on the level of automation applied in the process implementation. Various mechanisms can be employed to connect a SynerVision process to the environment. SynerVision process templates can invoke shell scripts and other executable programs or issue SoftBench BMS messages to communicate with other tools. External programs and tools can access SynerVision attribute data and manipulate SynerVision tasks only via SoftBench BMS messages. Product documentation provides an excellent tutorial on use of the tool for task management and simple template programming, but is not particularly useful as a detailed technical reference. HP justifiably classifies SynerVision as a "high support" product, meaning a typical user will require a great deal of support from the company to apply the tool effectively. Currently, HP's support organization is not adequately prepared to provide the necessary level of support. The engineering team has required assistance from SynerVision's developers to determine how to apply the tool, how to construct process templates, and how to circumvent problems. This technical assistance has been provided by HP as part of its commercial partnership with the Unisys STARS team, and is not available to the typical user.

3.2.3.1 Models of Use

HP describes four *models of use* for the product; we have employed three of the four to some extent in our implementation. *Managing Personal Task Lists* supports such things as managing simple *to-do* lists, listing and tracking time spent on monthly objectives, recording and annotating consulting time for different clients, planning and estimating work, and automating routine tasks such as running e-mail, printing reports, etc. *Managing Group Tasks* extends this idea to a team environment by allowing team members to share the task list, allowing a team leader to distribute tasks among the team members and coordinate dependencies via an internal task assignment mechanism. SynerVision supports these two models reasonably well and can automatically record time spent in each of the tasks if the performing individuals maintain the discipline to record each start/stop sequence in SynerVision.

While the external *presentation* of SynerVision capabilities suggests one of the *task list* models described above, its apparent application paradigm seems oriented more toward the *Developing Process Templates* model. In this model a SynerVision process programmer with a detailed process definition in hand uses a shell-like scripting language to implement a set of task hierarchies, present notes, messages, and dialog boxes to guide a user through the steps, and perhaps automatically initiate other shell processes which might invoke other tools, etc. Establishing this sort of capability is critical to our process integration efforts, however, SynerVision provides little support for the process engineer who must do this work. Work product and life-cycle modeling concepts are not apparent in any of SynerVision's capabilities. Inherent limitations in the use of interpretive shell programming, SynerVision's reliance on "convenience" functions to facilitate BMS messaging, and lack of control over the collection and display of system-level information inevitably necessitate development of low-level C routines to bridge the gaps. Thus, the process engineer must have expertise in the application of SynerVision, the SoftBench BMS, the SoftBench encapsulator, and traditional C development techniques to achieve reasonable process automation goals.

3.2.3.2 ChangeVision

SynerVision's 4th model of use, *Using SynerVision Process-Centered Environments* is represented by SynerVision's companion product ChangeVision. ChangeVision implements a software change control process based on an underlying tracking system (QualTrak's *Distributed Defect Tracking System (DDTs)*, required, not included) and HP's *Branch Validator*, which provides test coverage metrics (not required). Unfortunately, without DDTs or an equivalent tracking system, ChangeVision has no value whatsoever. We may, perhaps, gain some knowledge about integrating tools with SynerVision by examining the techniques used for ChangeVision, but otherwise ChangeVision cannot contribute to our environment development efforts at this time.

3.2.4 PCMS

PCMS is being applied to establish traditional configuration management (CM) and change control capabilities in our enactment environment, occupying the *Product Manager* role. A homegrown SoftBench encapsulation allows PCMS to receive BMS requests from the environment, instructing PCMS to create design *parts* within an overall *product* structure, associate component *items* with the parts, and populate the items with the contents of Unix *working files*. PostScript

and clear ASCII representations of a document are built automatically as *derived items* in the part structure, based on construction algorithms associated with the user-defined *item type*. BMS messages are also used to access PCMS-controlled items for modification and life-cycle management. Further development may allow PCMS to transmit similar messages back to the environment to communicate product information or request product support services.

The PCMS user interface is evolving as the product matures. Most administrative operations, control plan definition and role assignment, for example, are accessed via an Oracle Forms interface which is a residual artifact of early versions of the tool. The dependence of this interface on function keys makes application in a heterogeneous hardware environment difficult. A more user-friendly, though non-intuitive, X Window System interface is provided for more commonly used product manipulation functions (check-in, check-out, etc.). Duplicative implementation of complex menu structures and unusual terminology contribute to a general failure of the tool to communicate its intended model of use. Formal training for process engineers in use of the tool, definition of control plans, creation of product structures, etc. is a pre-requisite to its successful application. Consulting services are available from the developers to assist with these activities.

Product data is stored in an Oracle database, potentially supporting flexible SQL access. Implementation of the tool's capabilities in a PCTE (Portable Common Tool Environment) or other object-management-system-based environment has been investigated and found to be practical. A command line interface is provided for many operations, making it possible to populate product structures and manipulate controlled items through SoftBench messages, under the control of SynerVision. The command line interface has proven somewhat less useful in the extraction of product information. Use of an available C Application Programming Interface is under investigation. PCMS documentation provides an adequate reference to individual capabilities, but does little to guide a process engineer through the many operations that must be performed to implement a viable configuration management process.

PCMS was designed to support configuration management for software development projects. Its apparent application paradigm reflects the processes associated with management and control of software artifacts. The physical control of software source code modules used to generate object modules, which are then used to build a set of executable products is an ideal application for the tool. Direct support for classical configuration management objectives (identification, control, accounting, and auditing) is provided. Developmental CM activities at all levels of abstraction are supported (versioning, change control, product configuration, automated product generation, baseline management, release control, etc.). These capabilities can also be used to manage hardware, courseware, and documentation configurations to the extent that their development life-cycles are compatible with that of software. The inability to define a control plan that allows modification of an item after it has advanced beyond its initial state (see section 3.1.2.7) may restrict the applicability of PCMS to other kinds of processes.

3.2.5 Amadeus

Amadeus is a metrics collection, analysis, and reporting tool that can be applied in a stand-alone or integrated application. Amadeus can be invoked through a GUI, application program interface (API), command-line actions, or messages sent to it through HP's SoftBench BMS. Amadeus can

be run in manual mode (i.e., through GUI or user command-line actions), automated mode (i.e., Amadeus agents or command received from SoftBench or API), or a combination of both modes (e.g., data entry being applied through both the GUI, agent(s), and SoftBench BMS messages).

The training supplied by Amadeus Software Research (ASR) to date has consisted of two types: 1) introductory; and 2) advanced. The introductory course consisted of a comprehensive tool overview integrated with several laboratory tool usage sessions. The advanced course consisted of an overview of such tool features as designing and implementing agents and optimizing repositories, complemented with several laboratory tool usage sessions. The Amadeus training by ASR is outstanding and consider essential for successful tool application. The Amadeus tool documentation is still evolving and improving. It is not recommended that an organization rely solely upon Amadeus documentation for initial tool usage.

Data collected for analyses and for generation of reports and graphs by Amadeus is contained within an Amadeus repository. Repository data can be exported to other tools, imported into the repository from other tools, or entered from the GUI through an entry template. The Amadeus repository consists of records which can at maximum contain 30 fields. The repository can be optimized for space/time performance by reducing the number of fields in a repository record. The fields in a record can be assigned to unique data elements for an application. It is conjectured that assignment of fields to application data elements will be critical if repositories from different applications are to be merged in the future. The PEP project used the tool's preset field definitions.

All data processed (i.e., for collection, analyses, reports, and graphs) by Amadeus is implemented in templates, realized by a set of Amadeus programing constructs that can be assembled in the form of one of the following templates: 1) entry; 2) export; 3) import; and 4) graphs and reports. At present, our application is not substantial enough to reveal any shortcomings of the constructs. We do note that better documentation of the analyses constructs would greatly abet usage. There are 87 templates that are included in a baseline repository. Our findings indicate that in the majority of our applications, these templates can be modified for reuse. It is highly recommended that applications attempt to minimize the number of templates used, since their design, development, and testing can be quite costly in terms of time and effort. Amadeus programming constructs that support parameterization are needed to support template minimization. Although all templates developed for the PEP pilot are required to be documented, a standardized set of guidelines are needed to assist template developers in providing the appropriate level of documentation. We predict that as the Amadeus user community grows (i.e., as observed in the STARS Technology Transition Affiliates Amadeus Users group) standards and good practices will emerge and become common.

As implied above, the Amadeus process paradigm is based on collecting data, analyzing data, and generating reports and graphs. In addition, Amadeus provides automation features (for example agents that collect data based on file monitoring events) to facilitate metrics based applications. For the PEP pilot, the Amadeus process paradigm and features have been extremely useful. The effort and time experienced on the PEP pilot for using this tool are minimal, however, development of agents requires experienced script writers.

3.2.6 MSP & PM

During our initial information gathering and process analysis activities, a set of tools to help record, analyze, and maintain the consistency of our PBS information would have been useful. Even for our simple target process, the amount of textual information collected was difficult to manage; for more complex processes it would be virtually impossible without automated assistance. The *Minimally Structured Processes (MSP)* and *Process Mapper (PM)* tool sets were incorporated into our environment as research prototypes after the majority of our process analysis efforts had already been completed. The tools are currently being used to support process analysis activities for the Army STARS Demonstration Project. Early feedback from the Demo Project indicates that the tools provide the kind of automated support needed.

The MSP & PM tool sets are currently availabile only in conjunction with process engineering services provided by SDA. SDA has expressed an interest in supporting productization of the technology by commercial developers of process engineering support technologies, either as part of an integrated process engineering support environment, or as a separate product. As part of its overall commercialization strategy, the Unisys STARS Team is encouraging STARS Prime Affiliates and other commercial developers of process engineering support technologies to cooperate with SDA in this endeavor. Response to date has been favorable.

3.3 Encoding the Process

As described previously (see section 3.1 "Capturing the Existing Process") we found that, for our target process, attempting to encode the existing process as a simple hierarchy of steps was not viable. By simply reproducing the same sequence of detailed steps being executed manually, we created an unnecessary layer of computerized intervention between the user and the goal. In trial executions, detailed, low-level, sequentially dependent steps seemed to become less efficient when executed from within the hierarchy. It often took longer to tell SynerVision that the task had started and stopped than it did to perform the task. Some sequences could be readily executed via shell scripts invoked from within the task structure; others would have necessitated the development and presentation of complex information models to support the human agent's decision-making process. Absent significant levels of automation, SynerVision's potential to track and report task completion depended upon the discipline and diligence of its user.

An analysis of the purpose of the "as-is" process was initiated and a "to-be" process defined. Attempts to encode this new process for enactment were met with some frustration. The candidate tool set does not provide a cohesive means of encoding the activity, data, and behavioral information needed to execute the target process. Individual tools provide adequate capabilities to support their own process roles, but each employs its own data representation and instruction set. The resulting process implementation, though driven by a predefined process, was still largely centered around the human agent's invocation of individual tools. The cognitive load on the human agent and the potential for reversal of agent/support roles were significant.

3.4 Automating Process Steps

To realize the environment's potential to effectively manage the target process, automation of the interactions among the individual tools is necessary. SynerVision process *templates* provide one

mechanism for establishing programmatic control over the interactions. Based on the Unix Bourne shell, SynerVision's interpretive template programming language supports the definition of task structures, inter-task dependencies, and task-dependent actions. Manual and automated actions can execute template commands directly, invoke external shell scripts or other executables, or exchange BMS messages with other SoftBench-encapsulated tools.

Due to the wide array of interface capabilities available through SynerVision, and the relative convenience of shell programming, the team chose SynerVision templates to provide the middleware needed to coordinate execution of the process. The design approach involved isolating process control activity within the SynerVision templates as much as possible, transferring control to another tool or external program only when necessary. Invocation via BMS messages of individual tool capabilities (particularly within PCMS and Amadeus) was preferred over non-contextual invocation of a tool's graphical user interface.

The convenience of an interpretive language was offset to a great extent by problems associated with writing multi-level shell programs. SynerVision actions are interpreted by an instance of the shell separate from that used to interpret the main body of a process template. SynerVision employs the shell-based convenience functions of the SoftBench *ciclient* utility for exchange of messages with the BMS. PCMS and Amadeus command strings, passed via BMS messages, are interpreted by yet another shell. The SoftBench *Encapsulator* was used to present the human process agent with a data entry form similar to a printed form employed in the manual process implementation. Transfer of data between the shell-like encapsulation program and the SynerVision template was problematic. In combination, the hierarchy of shells interpreting template information, particularly shell variables and strings, proved difficult, though not impossible to manage. For trivial SynerVision actions, such as invoking tools, executing simple shell commands, transmitting BMS messages to which no response is expected, or performing other operations under the complete control of SynerVision, the template language proved adequate. For more complex interactions, involving the exchange of dynamic process or product information among tools or the transfer of process control from one tool to another, interpretive programming has proven wholly inappropriate.

The level of programming expertise required and the amount of design coordination needed to ensure consistency among the disjoint interpretive programs is equivalent to that of traditional software development methods without the benefit of access to system functions and data structures normally available through C or other high-order languages. The tendency of tool developers to support simple interpretive programming is understood and such support may be necessary, however, the corresponding tendency to forego support for more structured, formal, software development techniques negatively impacts the utility of the tools. Implementation of a fully functional high-order process programming language, either as part of a particular tool's capabilities or as a separate product would be welcomed. This, combined with an effective object management system to support integration of process and product data, could stimulate a significant advance in the state of process automation practice.

4.0 Continuing Experimentation

The next phase of the Process Enactment Pilot project will examine the ability of the integrated tool set to support live performance of the *Deliver STARS Document* process. Planning activities will address formal adoption of the re-engineered process, establishment of measurement goals for the project, and migration of the new process into the department's operating environment. Unresolved technical issues will be addressed in the context of their impact on the ability of the human agent to perform the process and of the process engineer to maintain it. Where significant technical obstacles exist, alternative approaches to enacting the process will be examined.

4.1 Executing the Process

For a period of approximately one month, following management acceptance of the re-engineered process, the team will experiment with execution of the process using documents already under CM control. During this time the process may be revised to facilitate ease of use or to correct errors. Absence of a fully functional PCMS message interface may necessitate exclusion of PCMS from the tool set until appropriate interface techniques can be devised. Upon successful completion of the trial use effort, the automated process will be employed as the primary mechanism for delivering STARS documents.

4.2 Evaluating the Process Definition

The Amadeus measurement system will be used to record basic information about contract deliverables, projected lead times, etc. for analysis with actual process performance data. Automated analysis of such data to report potential impact on schedule and resources will be implemented as resource allocations allow. Product measurements (for example, document size, sentence length, percentage of white space, etc.) will be collected to exercise PCMS-Amadeus communications. A set of standard process events (BEGIN/END, BEGIN/END-SETUP, BEGIN/END-COMMUNICATION, BEGIN/END-WRAPUP, BEGIN-SEND, BEGIN-RECEIVE, END-NORMAL, END-ABORT) will be recorded to facilitate computation of various elapsed time intervals for potential correlation to resource and schedule data. Other project, product, and process measurements will be recorded as necessary to support measurement goals.

4.3 Adapting the Process

As the process implementation matures, whether within the context of the Process Enactment Pilot project or not, further automation will be encouraged. Additional support for preparation of letters of transmittal, delivery notices, and other *companion work products* will ultimately be required, as will automation of the record-keeping functions normally attributed to the routing sheets and tracking forms used to support the process. As the tool set and enactment technologies mature, as the proficiency of the process engineering team increases, and as analysis of metric data exposes areas of potential risk, it may be desirable to revise, refine, or replace various aspects of the implementation to take advantage of the advances.

Data Reference: STARS-VC-A023/009/00
INFORMAL TECHNICAL REPORT
Process Enactment Pilot Project
Phase II
Lessons Learned

Appendix A: Notes on Tool Usage

1.0 AutoPLAN

1.1 User Interface

The user interface is generally consistent, though not always intuitive. All operations are accessible via pull-down tool bar menus which invoke various kinds of dialog boxes; tool bar icons are provided for alternate access to some of the more commonly used features; and a *Canvas Menu*, similar (but not identical) to the tool bar *Edit* menu pops up when the user "right-clicks" within the body of a chart. In some cases the use of dialog boxes is logical; in others a simple nested menu structure would be more appropriate. With a little experience most of the user interface quirks become minor distractions; with prolonged use they could become major irritants.

1.2 Producing Schedules

We tried to generate separate schedules for each member of the team by generating filtered views of the master GANTT chart. For some reason, AutoPLAN does not allow indented task structures (as used in all GANTT charts) to be filtered; the indentation must first be temporarily disabled. Furthermore, filtered views are automatically resorted in order of their internal activity codes. Various sorting options are available, but there is no option to sort "as entered", and it has not always been possible to combine sorting options to achieve the desired results. The combination of these phenomena restricts the utility of the GANTT charts.

1.3 Reporting

Initial attempts to use AutoPLAN's reporting capabilities were disappointing. A menu is provided for customizing the format of a printed report. Unfortunately, not all of the settings are saved along with the chart (most notably, the *Report Title*), making report generation more tedious than need be. Producing a printable GANTT chart is frustrating, requiring trial-and-error manipulation of both, the screen image, and the report format to make the inevitable multi-page print-outs readable. Production of readable print representations of almost any diagram, especially CPM diagrams and Work Breakdown Structures, was highly dependent on manual cut-and-paste operations.

1.4 Resource Analysis

We ran several resource analysis reports in an attempt to detect labor overloads. Initial results exposed an apparent anomaly in AutoPLAN's calculation of resource distributions; the reports did not accurately total resource data for *summary* tasks. We have attributed the problem to a difference between the tool's application paradigm and that of the user. Our typical practice in building a project structure is to enter a set of high-level tasks, estimate the duration of the tasks and/or effort required to complete them, assign available resources, then decompose the tasks. When a task is decomposed, AutoPLAN automatically redefines it as a parent, or *summary*, task whose scheduled duration is determined by the scheduling parameters of its children. However, effort and resource attributes are not automatically allocated to the task's children, so the data must be re-entered for each child. If resource allocations for the parent tasks are not manually removed, when AutoPLAN computes effort and resource distributions, the values originally associated with the parent task are accumulated with those of its children, effectively doubling the computed totals. In deeply nested project structures, the cumulative effects of this anomaly are devastating. It is relatively simple to avoid the anomaly by waiting until all tasks have been fully decomposed before entering effort and resource data. We would prefer not to be forced to alter our project planning paradigm in this way.

1.5 Reliability

The tool has crashed on a couple of occasions. Although the specific cause has not been investigated thoroughly, it appears to have been related to the designation of an *imposed date* for a milestone. The date of a review was fixed by imposing the start date for the milestone. Dependencies were established between the milestone and the activities preceding it. The computed duration for the lead-in activities may have extended beyond the imposed milestone date. When asked to calculate a schedule, AutoPLAN dumped core.

2.0 Design/IDEF

2.1 User Interface

The illogical organization of Design/IDEF's pull-down menus inhibits productive use of the tool. After a year of frequent use, it can still take several minutes to locate menu operations.

2.2 Printing Diagrams

The user must decide whether to print to a PostScript file or to a printer prior to tool invocation; the selection is made via definition of 2 environment variables, one to specify use of the Unix "echo" command for printing, and the other to identify the directory into which the output file is to be written. The user is provided no means of specifying the name of the output file. The tool's output file name generator is designed to ensure uniqueness, not to reflect information about the model being printed, so renaming is almost always necessary. The PostScript produced is not encapsulated, making it cumbersome to incorporate diagrams into other documentation.

2.3 Data Accessibility

The tool is effectively closed to its environment, providing no external access to its database. Access is provided only at the model level, with a complete model represented in a single Unix file.

2.4 Process Support

The tool is effective at drawing activity diagrams, but provides no support for author/reader cycles, version management, or other important aspects of the IDEF0(SADT) method. This effectively limits its utility to that of a general purpose drawing package tool. With more methodological support, either within the tool or through integration with other technologies, Design/IDEF could qualify as a process engineering CASE tool.

2.5 Reliability

The Unix version of the tool is unreliable, unpredictable, and receives minimal developmental attention from the company. Frequent core dumps have occurred in association with the use of embedded text links, making this important feature of the tool unusable. It is altogether risky to place much trust in its operation, but until other SADT modeling tools are available on Unix platforms, the risk to our relatively small project is tolerable.

3.0 SynerVision

3.1 User Interface

A simple task hierarchy like that employed by SynerVision is not a fully effective means of describing a process. SynerVision provides various mechanisms for enhancing its hierarchical process representation; these seem better suited to support the *execution* of a process than its *definition*. Though the user interface for each of these capabilities has an X Motif *look*, their *feel* is distinctly menu-oriented, and there is little consistency in the way they operate. Manipulation of display objects is accomplished through button clicks that invoke various menus; many of the operations thus performed should not require accessing an intermediate menu.

SynerVision's *Project*, *Task*, *Attributes*, *Navigate*, and *View* menus are identical for all tasks and are accessed via the tool bar. A single unnamed menu provides access to several unrelated operations: starting and stopping execution of a task, manually changing the status of the task, and relocating the task in the hierarchy. This menu is accessed by clicking the right mouse button on the target task. The *Actions* menu, which provides access to manual actions specific to a particular task, appears in the tool bar, along with the non-task-specific menus. One might expect all menus that apply to all tasks equally to reside in the tool bar at the top of the SynerVision window, and menus tailored to specific tasks to be associated with the task, or at least to see consistency in their placement.

Double-clicking on a task title causes SynerVision to re-orient its display, focused on that task; execution of the task would seem a more logical choice of operations for "double-click", and would be consistent with other X environments with which we are familiar. HP's own VUE windowing environment allows the user to define the action associated with a "double-click" on a particular class of display object; a similar capability would enhance SynerVision's usability.

Moving tasks around in the hierarchy is relatively simple, though it requires a 6-step drag and drop sequence. The operation should require no more than half that number of steps. Establishing dependencies between tasks uses a different, more cumbersome interface. Task attributes are divided into groups called *Notes*, *Dependencies*, *Access*, *Actions*, and the cryptic *Basic*, which actually includes scheduling information. Alternative access to these and more attributes is provided through a *Low Level Attributes* group. Each group is manipulated through a completely different interface. Support for defining trivial manual actions is adequate, but more complex manual actions and all automated actions must be defined via a Unix shell interface. Manual actions can be inherited by the children of the task that owns the definition of the action; automated actions cannot.

3.2 Models of Use

HP describes four *models of use* for the product. *Managing Personal Task Lists* supports such things as managing simple *to-do* lists, listing and tracking time spent on monthly objectives, recording and annotating consulting time for different clients, planning and estimating work, and automating routine tasks such as running e-mail, printing reports, etc. *Managing Group Tasks* extends this idea to a team environment by allowing team members to share the task list, allowing a team leader to distribute tasks among the team members and coordinate dependencies via an

internal task assignment mechanism. SynerVision supports these two models reasonably well and can automatically record time spent in each of the tasks if the performing individuals maintain the discipline to record each start/stop sequence in SynerVision.

While the external *presentation* of SynerVision capabilities suggests one of the *task list* models described above, its apparent application paradigm seems oriented more toward the *Developing Process Templates* model. In this model a SynerVision process programmer with a detailed process definition in hand would use a shell-like scripting language to implement a set of task hierarchies, use notes, messages, and dialog boxes to guide a user through the steps, and perhaps automatically initiate other shell processes which might invoke other tools, etc. Establishing this sort of capability is critical to our process integration efforts, however, SynerVision provides little support for the process engineer who must do this work. Work product and life-cycle modeling concepts are not apparent in any of SynerVision's capabilities. Inherent limitations in the use of interpretive shell programming, implementation of "convenience" functions to facilitate BMS messaging, and lack of control over the collection and display of system-level information inevitably necessitate development of low-level C routines to bridge the gaps. Thus, the process engineer must have expertise in the application of SynerVision, the SoftBench BMS, the SoftBench encapsulator, and traditional C development techniques to achieve reasonable process automation goals.

3.3 Writing Process Templates

SynerVision provides little direct support for the *Developing SynerVision Templates* model. Essentially, the process engineer must apply traditional software engineering techniques to construct an interpretive "Bourne-shell-like" program to create and manipulate SynerVision tasks, their attributes and actions. It is possible to take advantage of tool features to reduce the tedium somewhat, but this should not be considered the primary mode of operation. A task hierarchy can be entered via the interactive outliner used for task management. Sequential dependencies between tasks, based on changes in task status, can be defined. Manual actions can be defined and tested for tasks or groups of tasks in the hierarchy, and notes can be entered. It is possible to define automated actions via the *Low Level Attributes* menu, but the user interface for doing so is virtually incomprehensible and its use is unadvisable. The task structure, including all dependencies, actions, notes, and static attributes can be translated to SynerVision template format, using the *Task/Generate Template* operation on the tool bar or the *gen_template* shell script.

This method may be useful in a process prototyping environment when the process is changing rapidly and small revisions need to be tested quickly. Use of the interactive outliner to enter the initial task structure can also provide some benefit to those who prefer point-and-click operations to text editing, even though most operations seem to require too much mouse manipulation. For personal task hierarchies, static processes that are identical in each instance, or templates for which maintenance is uncomplicated, the method may prove adequate. For development of reusable templates, templates which are placed under formal change control, structured sets of templates, or incorporation of automated actions, some off-line revision of the template will be necessary. Most such revisions are retained when the template is re-instantiated as a SynerVision process, but are lost if the template is re-generated from SynerVision, making the method non-iterative.

3.3.1 Language Considerations

Interpretive shell languages are an integral part of any Unix environment; Bourne shell is, perhaps, the most common. It provides an appropriate mechanism for programming trivial applications, but does not adequately support development of complex process programs. Its commonality among hardware platforms makes it a logical choice for programming simple interactions between a single tool and the Unix environment. When used to coordinate operations among several such tools, communicating through multiple layers of shells, its utility degrades quickly. Even determination of appropriate string quoting sequences to support transfer of data through multiple shells can require a great deal of trial and error.

SynerVision complicates this situation further by employing multiple shells internally. Manual and automated actions are interpreted by a different instance of the shell than the main body of a template, placing a communication barrier between the two (we have not located this information in any SynerVision documentation). The product's developers have advised us that the quoting rules are those of the Bourne shell, except that the contents of actions are sent through a second shell at action execution time, and "an extra level of quoting is sometimes required". If the termination tag for the *here document* that defines an action, is not *quoted*, variable substitution, interpretation of back-quotes, etc. is performed when the action is defined - that is, when the template is instantiated, otherwise these operations are deferred until the action is executed. Depending on the combination of quotes used in the termination tag and used within the action definition, the programmer's ability to manipulate environment variables and task attributes (see section 3.3.2 "Data Storage") can vary greatly. A working combination can probably be found for most applications, given enough experience with the tool. Development or adoption of a fully functional process programming language is needed.

3.3.2 Data Storage

Due to inadequacies of shell-based programs in storing persistent data, alternate storage mechanisms are sometimes needed. SynerVision provides for definition of custom attributes for storage of user data associated with tasks. The additional attributes must be defined in the database schema for an entire project and every task carries the burden for storage of the data. This method is appropriate if one assumes all attributes apply equally to all tasks (attributes like *project name*, *task owner*, and *elapsed time* are good examples). For storing information pertinent only to one kind of task, burdening the rest of the task hierarchy with the storage of meaningless attributes is unacceptable. In actual practice, empty attributes consume only minimal storage space, but the time required to load them into memory can be significant, especially for large task hierarchies. To achieve reasonable flexibility, we have resorted to storage of such data in an external database, which is wholly undesirable.

3.3.3 Retrieving User-Defined Data

SynerVision provides a mechanism (*svprompt*) for prompting the user for information. Its use is restricted to the entry of a single filepath, a single- or multiple-line text string, and the selection of exclusive or non-exclusive toggle buttons. For manual process flow control, or entry of a single data item, the function performs well. Our application requires the retrieval of several data items (CDRL reference, author, location of working files, etc.) for each instance of the target process;

we employed the *SoftBench Encapsulator* to build a data entry form which is presented automatically to the user when the process is initially executed. The Encapsulator provided no mechanism for passing the data back to SynerVision, but we were able to revise the SynerVision template to retrieve the data via output to a temporary file; by restricting the format of string entries, we were also able to retrieve data directly into the template shell. Addition of a data entry mechanism to SynerVision capabilities would be welcomed.

3.4 Reporting

SynerVision's reporting capabilities are rudimentary. It is possible to print a copy of the task structure as it appears on the screen, but there is no capability for customizing the format of the resulting print. Bourne shell scripts can be written to produce reports in any format desired, but expanded support from the tool is needed.

3.5 Documentation

The documentation set for SynerVision, when read from beginning to end, presents an excellent tutorial on use of the product's basic capabilities. The index provides reasonable access to descriptions of individual capabilities. On-line help duplicates the information in a well-presented point-and-click browser. As a technical reference the documentation is less useful. Unix "man" pages are included in the printed material to augment the level of technical detail. The complexities of building an interface between SynerVision and its environment (appropriate use of SoftBench messages, coordinating shell interactions, using *svprompt*, etc.) need to be addressed in the documentation.

3.6 Support

HP classifies SynerVision as a "high support product", meaning the company expects users to need assistance in its application. The individuals who provide support via the HP Customer Response Centers are cooperative and willing to serve, but their knowledge of the product is unacceptably limited (we have found this to be the case with most other HP products, as well). An electronic reporting mechanism is available, providing a mechanism for formally documenting problems and questions (handled by Response Center staff), as well as access to general product information and user comments. The developing organization has helped to overcome this deficiency by providing technical assistance when necessary, but overall the support system doesn't work well. Problems reported to the Response Centers via telephone are not usually tractable via the electronic database, and vice versa. Organizational problems affecting communication and coordination of response are apparent.

3.7 Miscellaneous

3.7.1 Core Dump

For months, whenever we attempted to bring up a Graphical display of a SynerVision process, the tool would crash. HP support could offer no assistance in solving this problem. The cause turned out to be the existence of a definition for X resource (*cursor) in the user's environment. The problem was solved by removing the *cursor definition from the user's X resource defaults, unnecessarily restricting local control of the user's work environment.

3.7.2 Deleting Tasks from Shared Projects

Tasks cannot be deleted from shared projects, which introduces some difficulty during experimentation and debugging. HP support suggested moving the task to the owner's *personal project* and deleting from there; it worked.

3.7.3 Multiple Actions

Only a single action matching all the discriminators (*Status*, *New*, *Inprogress*, *execute*) may be defined for a single task. HP support suggested including multiple shell script calls in a single action to split things up, which did not product the desired effect.

3.7.4 Sparc Support

SynerVision creates and manipulates its own directory structures within the user's task repository, and employs a file locking mechanism (*pmlockd*) to ensure file integrity. File organization on SUN workstations is different, so a different version of *pmlockd* is needed. HP promises development of this utility for the Solaris OS, but has no plans to support SunOS. Attempts to license the source code for development of a SunOS implementation have been unsuccessful.

3.8 Summary

SynerVision is a significant new technology whose potential has not been realized. The SynerVision concept is viable, but its implementation falls far short of our expectations. Preliminary observations indicate that SynerVision will perform well purely as an engine for execution of an automated process. At present, the ability to automate such execution may be SynerVision's only advantage over other, less complex, less expensive software packages. To take advantage of that ability, additional capabilities are needed (within the SynerVision product, or elsewhere) to support the process engineering effort required to develop executable process templates. Current template development support is highly dependent on other HP products with which SynerVision has not been fully integrated, perhaps due to a failure of the developers to anticipate complex application of the product. By thoroughly re-examining the product's multi-level objectives and by investing in a significant re-engineering effort SynerVision could become a valuable component of a process-driven environment within the next 2-5 years. Until such an effort can be completed, SynerVision's effort/benefit ratio will remain quite high, and our confidence in recommending the product for operational use will be limited.

4.0 PCMS

PCMS was designed to support Configuration Management for software development projects. As such, its application paradigm reflects the processes associated with management and control of software artifacts. The physical control of software source code modules used to generate object modules, which are then used to build a set of executable products is an ideal application for the tool. Direct support for classical configuration management objectives (identification, control, accounting, and auditing) is provided. Developmental CM activities at all levels of abstraction are supported (versioning, change control, product configuration, automated product generation, baseline management, release control, etc.). These capabilities can also be used to manage hardware, courseware, and documentation configurations to the extent that their development life-cycles are compatible with that of software.

4.1 User Interface

The complexities and relative immaturity of PCMS tool necessitate formal training, especially for process engineers or other individuals responsible for integration and application of the tool. Different capabilities of the tool employ different user interface techniques, and reflect a European language influence on terminology that is often confusing. Addition of an interactive help facility would be a great benefit to the product.

In early versions of the product, user interaction with PCMS was provided via an Oracle Forms interface (PCMS employs Oracle as its underlying database). In PCMS version 4.0, an X/Motif interface (XPCMS) has been included for the first time. Most operations performed by a typical user (creating parts hierarchy, checking items in and out, etc.) are accessible via the X interface. Other less frequently used, but nonetheless critical, functions still rely on Oracle Forms. PCMS setup (creation of the control plan) must be done through Oracle Forms. The forms interface is relatively difficult to use, due to its pervasive use of control key combinations and function keys. Key sequences for many commands vary significantly, depending on the configuration of the keyboard being used. Product documentation only reflects a small number of different keyboard configurations, and is not reliable. In a heterogeneous environment populated with workstations from many different vendors, this sort of dependency is unacceptable. SQL is planning to completely replace the Oracle Forms interface with the X/Motif interface in a future version of PCMS.

Annoying inconsistencies in the Forms interface have been noted. For example, the dialog box used to prompt the user to COMMIT changes before exiting the tool is not always displayed. Both the XPCMS interface and the forms interface can be opened from different windows at the same time to allow frequent iteration between the two when testing a newly created part of the control plan. It is sometimes necessary to refresh the XPCMS screen, but this a simple operation. PCMS utilizes the Unix sendmail system to notify appropriate users when the life-cycle state of a controlled item has changed. During PEP experimentation with PCMS, the number of messages transmitted has been an annoyance. In our process enactment, we would also like to use a different mechanism to communicate this kind of information, but we have been unable to disable the feature.

PCMS also supports a command line interface to many of its functions. The command syntax is tricky and related documentation is not well organized. We have been somewhat successful in the

application of the command line interface as the basis for SoftBench access to the tool. It has proven relatively simple to use the command line interface to create and manipulate product structures within the PCMS database, but we have been unable to identify suitable techniques for querying the structures to find the items associated with a part, etc. It seems such information can only be accessed interactively or through the C Application Programming Interface. This difficulty places severe restrictions on our ability to use the tool in an automated process enactment. We have also been unable to route PCMS error messages to SynerVision for visual presentation. We expect to circumvent some of these difficulties as we gain experience with the tool.

4.2 Control Plans

PCMS provides a degree of process control through the use of a *control plan.* This control plan is used to create roles which control PCMS user access to groups of items (files under configuration control) and control promotion of items through their life-cycle states. Life-cycles for item types, design part types, rules for design part types, change management rules, template forms, and attributes of items and design parts. Design parts are used to create a hierarchy for management of items. A control plan must be defined before any instance of an item type can be stored. The control plan is set up using the Oracle Forms interface to PCMS; access via XPCMS is planned for a future release. There is no mechanism in PCMS for creating a printed report describing a control plan.

Development of a PCMS control plan is difficult for the first-time PCMS user. PCMS training classes include a discussion of how to enter the individual sections of the control plan, but do give insight as to the strict sequence in which these sections must be entered. The documentation does not provide more guidance. Within the sections of the control plan, it was difficult to determine what we needed to enter in specific fields. There was also little guidance available about which sections of the control plan and fields within these sections were mandatory. Different sections of the control plan often referred to the same field by different names. A more comprehensive training program and documentation and a good help that guides the creation of a control plan would be of great benefit. Even PCMS users need some training about the control plan, since this plan governs what is allowed in their interactions with PCMS.

Designed for management of software source code, PCMS control plans include state life-cycles for item types, with roles for each life-cycle transition. The authority to change states for an item version is rigidly restricted by these life-cycles, so care must be taken to ensure that the level of control specified is compatible with practical operations. Only one user can be authorized to change the control plan for a given PCMS *product.* (Within PCMS a *product* refers to an entire design part hierarchy.) For some types of managed items, strict configuration management may be necessary to ensure that the reasons for each change in a product are known, and to ensure recovery to a previous configuration. Other types may require minimization of life-cycle restrictions.

PCMS item life-cycles allow an item version to change states based on degree of validation that has been performed on the item version. Regardless of the control plan defined for a particular item type, PCMS only allows modification of a particular version of an item while the version is in its initial state. This restriction makes it impossible to define a product life-cycle that supports incorporation of review comments, correction of typographical errors, addition of electronic sig-

natures, or any other modification without re-executing the entire validation life-cycle. Once a change is made, the item's life-cycle must be re-executed from the beginning. In a software development process, where any change to source code might necessitate re-execution of a life-cycle that involves code review, unit test, etc., the restriction may make sense. In other processes the restriction make it impossible to accurately represent the true life-cycle of a the process's work products using PCMS item life-cycles. The initial state for changes restriction also makes it impossible to give different roles item change authority at different points in the item's life-cycle. We also found the necessity to define control plans early a hindrance to prototyping and debugging PCMS product structures, etc. The ability to disable control plan enforcement during such activities is desired.

Life-cycles of items in PCMS are used to determine validation state of individual versions of items. Life-cycles can only be linked between a change document and a product, and these links are very inflexible. The PEP Project team created general life-cycle models for process work products. There is no mechanism in PCMS for entering these general item life-cycles and linking them to life-cycles of other items.

4.3 Product Hierarchies

PCMS allows the creation of hierarchies of *design parts*. Items are stored and controlled within this hierarchy. It was not trivial to decide how to best set up this design part hierarchy to match the needs of the PEP Project. Design part hierarchy should be part of the general product model and should be determined when creating this model, but certain choices make it easier to set up other PCMS item control functions such as baselines and configuration builds. Design part hierarchies can be printed in PCMS reports. It is worthwhile to note that the representations used to establish design part hierarchies and life-cycles in PCMS can also serve as adequate, if tool-influenced, descriptive models of the products to be managed.

4.4 Repository Construction and Access

PCMS stores control plan information, design part hierarchy, and item names within an Oracle database. Individual items are stored within Unix, with all item of the same type stored in the same directory. This directory structure necessitates creating unique names for items within a type. When items are extracted, these unique names are used instead of the original file names. This caused difficulties on the PEP Project, because external build routines assumed the original file names.

PCMS also allows the storage of item versions by storing the differences between these items (using SCCS), to save space. However, to use this feature every PCMS user must have write access to the Unix directory structure holding the items. Any user could circumvent PCMS by directly accessing this Unix directory structure.

4.5 Use of Environment Items

Environment items are used in PCMS to relate items for creating baselines and configuration builds. Different types of environment items are needed for baselines versus configuration builds, so duplicate environment items must be created to handle both cases.

4.6 Version Management

PCMS follows a policy of strict configuration control of all items. This policy cannot be circumvented. For example, an item cannot be deleted until every baseline and configuration build that uses the item is deleted. On the PEP Project, this policy made it difficult to represent parts of our process in PCMS, because we were allowing a more flexible policy.

4.7 Baseline Management

Baseline management in PCMS allows the creation of a baseline of a group of items. The baseline management facilities provided in PCMS made it easy to create baseline plans and to execute baselines.

4.8 Configuration Builds

Configuration builds are used in PCMS to create derived items, such as object files and executables from source code. Once the configuration build plans are created, executing these builds within PCMS is simple. Creation of the configuration build plans is very complex. Training is provided by SQL on configuration builds, and this training is recommended.

4.9 Additional Capabilities

The following capabilities of PCMS have not been exercised by the PEP Project to any significant degree, due to resource restrictions. These capabilities may be explored in the future.

- Design part and item attributes
- Change management
- Variants
- Release generation and control
- Archival, retrieval, and transfer

4.10 Documentation and Training

PCMS documentation and training was found to be inadequate for our needs. PCMS documentation provided information on each section of the Oracle Forms interface and each XPCMS command. No training or documentation was provided on the methodology that must be used create the PCMS control plan or on the methodology that must be used to create a product model to support that control plan. PCMS consulting is available to assist organizations in these areas.

4.11 Reliability

Both the PCMS Oracle Forms interface and the XPCMS interface were found to be fairly reliable. The XPCMS interface did crash and core dump a few times during our use of the tool, but in all cases the tool could be restarted. We were not able to determine what caused these crashes, since entering the same commands after the tool was restarted did not cause the crash to reoccur.

4.12 Tool Integration

SynerVision's hierarchical process was difficult to reconcile with PCMS's "circular" item version life-cycles, where item changes require cycling back to the original life-cycle state or creation of a new version.

4.13 Summary

PCMS is a suitable product for providing configuration control on larger software development projects. It includes direct support for the configuration management objectives of identification, control, accounting, and auditing. Setting up the PCMS control plan and structure is difficult for a PCMS novice, but consulting support can be obtained. Some difficulties were encountered in trying to adapt PCMS for general product support within a defined process. We will continue investigating whether PCMS can be used to provide this process support.